Static analysis and query optimization in semantic web data

Theivapulendra E.

Overview

1. Introduction
2. Basics of RDF and SPARQL with algebra
3. Pattern trees and Query Plans
4. Query containment
5. Enumeration and counting of well-designed SPARQL

1.Introduction

- **Semantic web data**
  - Data sources are incomplete
  - Users interested in partial answers
  - Large datasets are being made available due to the rapid emergence of linked data
  - Queries are executed in remote end points with sparql

- **Static analysis** is the fundamental task for **query optimization**
  - Containment
  - Equivalence

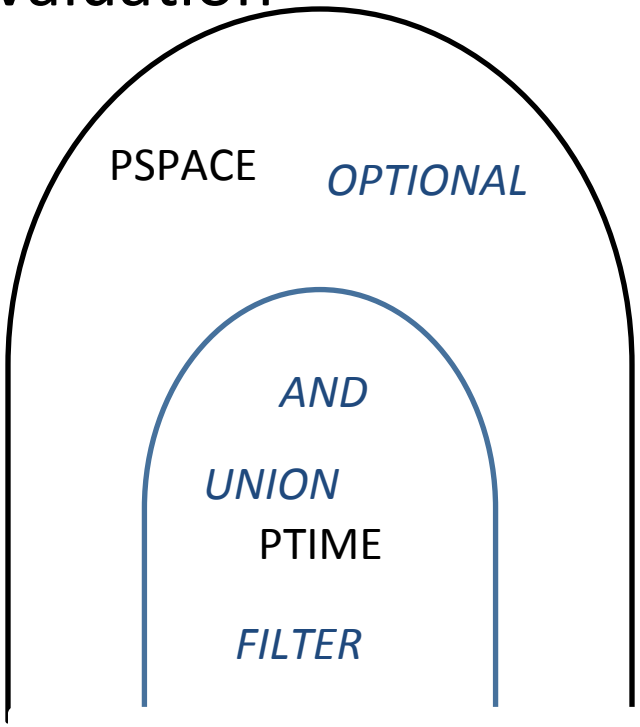
Previous related work

- Most works focused only in Conjunctive Queries and this research focusing on optional matching feature.

- Query optimization handled with query rewriting based on properties of operator and here it is done with relational algebra.

Why Optional matching feature?

- Complexity of query evaluation



- OPTIONAL alone leads to PSAPCE complexity in query evaluation
- In DBPedia more than 45% of queries use OPT operator.

2.Basics of RDF and SPARQL

- Considering only the ground RDF graphs
  - No blank nodes
  - No distinction between URI and literals
    - Thus, RDF triple pattern is a tuple in $U*U*U$
    - Active domain of a RDF graph G, $dom(G) \subseteq U$

2.1 Graph Patterns

- A triple pattern is a graph pattern
- If P1 and P2 are graph patterns, then (P1 and P2) , (P1 OPT P2) are graph patterns.
- Mapping in graph pattern
  - $\mu : V \rightarrow U$

  a triple pattern t and a mapping μ such that

  vars(t) ⊆ dom(μ),

  we denote by μ(t) the RDF triple obtained by replacing the variables in t according to μ

Compatible mappings

Definition

The mappings μ_1, μ_2 are compatibles iff they agree in their shared variables:

- $\mu1(?X) = \mu2(?X)$ for every $?X \in dom(\mu_1) \cap dom(\mu_2)$.
- $\mu_1 \cup \mu_2$ is also a mapping.
- $\mu_\varnothing = \{\}$ is compatible with every mapping.

Example

	?X	?Y	?U	?V
μ_1	R1	John		
μ_2	R1		J@edu.ex	
μ_3			p@edu.ex	R2
$\mu_1 \cup \mu_2$	R1	John	j@edu,ex	

Sets of mappings and operations

Let M1 and M2 be sets of mappings:

Definition

Join : $M1 \bowtie M2 = \{\mu1 \cup \mu2 \mid \mu1 \in M1, \mu2 \in M2 \text{ and } \mu1 \sim \mu2\}$

will be used to define AND

Left outer join: $M1 \mathbin{⟕} M2 = (M1 \bowtie M2) \cup \{\mu \in M1 \mid \forall \mu \in M2 : \mu1 \sim \mu2\}$

Will be used to define OPT

Semantics of general graph patterns

Definition

Given a graph G the evaluation of a pattern is recursively defined

- $[[(P_1 \text{ AND } P_2)]]_G = [[P_1]]_G \bowtie [[P_2]]_G$
- $[[(P_1 \text{ OPT } P_2)]]_G = [[P_1]]_G \mathbin{⟕} [[P_2]]_G$

Example

G : (R1, name, john) (R2, name, paul) (R3, name, ringo)
(R1, email, J@ed.ex) (R3, email, R@ed.ex)
(R3, web, ww.ringo.com)

Evaluate $[[\{(?X, \text{name}, ?N)\} \text{ OPT } \{(?X, \text{email}, ?E)\}]]_G$

$[[\{(?X, \text{name}, ?N)\}]]_G$ $[[\{(?X, \text{email}, ?E)\}]]_G$

Example

	?X	?N
μ_1	R1	John
μ_2	R2	Paul
μ_3	R3	ringo

$\bowtie$

	?X	?E
μ_4	R1	j@edx.ex
μ_5	R2	r@edx.ex

	?X	?N	?E
$\mu_{1U}\,\mu_4$	R1	John	j@edx.ex
$\mu_{2U}\mu_5$	R2	Paul	r@edx.ex
μ_3	R3	ringo	

Semantics of graph patterns OPT clause

- Graph storing information about professors in a university with following triples
  - (R1, name ,paul)
  - (R2, name, john)
  - (R3, name, george)
  - (R4, name, ringo)

(R1, phone, 777-3426)

(R2, email, john@acd.edu)

(R3, webPage, www.george.edu)

(R4, email, ringo@acd.edu)

(R4,webPage,www.starr.edu)

(R4, phone, 888-4537)

Evaluate Pattern over Graph-1

- Pattern P1=(**((?A ,name, ?N) OPT (?A email ?E))** OPT **(?A webpage ?W)**)

$[P1]_G$ =

	?A	?N	?E	?W
μ_1	R1	paul		
μ_2	R2	john	john@acd.edu	
μ_3	R3	george		www.george.edu
μ_4	R4	ringo	ringo@acd.edu	www.starr.edu

Evaluate Pattern over Graph-2

- Pattern P2=(**(?A ,name, ?N)** OPT **((?A email ?E) OPT (?A webpage ?W))**)

$[P2]_G$ =

	?A	?N	?E	?W
μ_1	R1	paul		
μ_2	R2	john	john@acd.edu	
μ_3	R3	george		
μ_4	R4	ringo	ringo@acd.edu	ww.starr.edu

$[P1]_G \neq [P2]_G$

So $[[((A \text{ OPT } B) \text{ OPT } C)]]_G \neq [[(A \text{ OPT } (B \text{ OPT } C))]]_G$

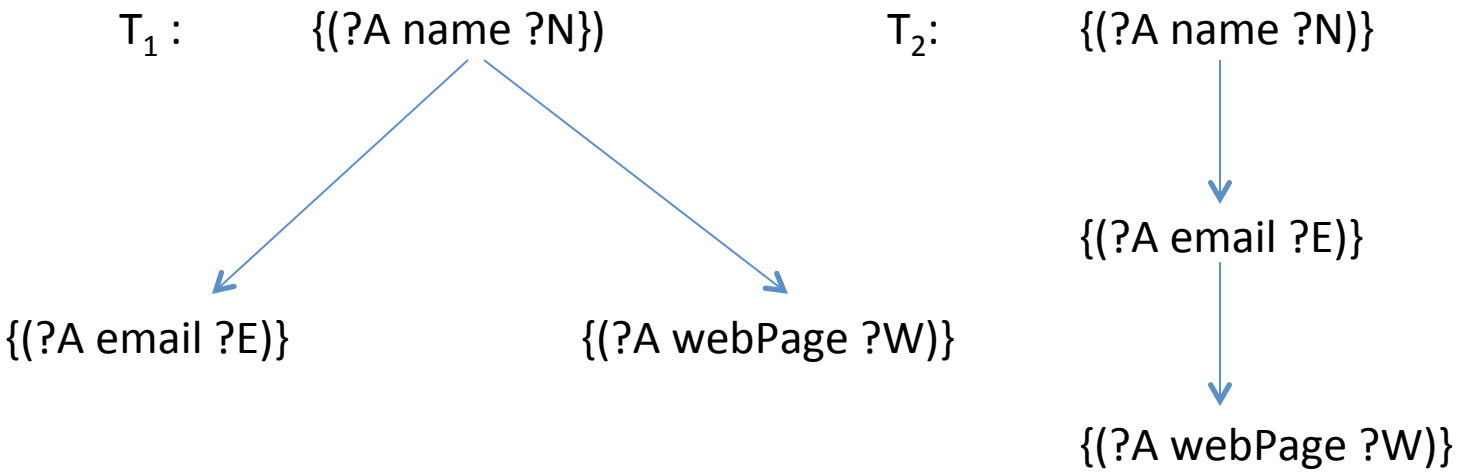
Well-designed graph patterns

- A pattern P is well-designed if for every sub pattern p'= (P1 OPT P2) of P every variable ?X occurring in P, it holds that :

  if ?X occurs inside P2 and outside P', then ?X occurs inside P1.

  Ex: P= ((**?A** name ?N) OPT ((?A email ?E) OPT (**?A** webPage ?W)))

3. Pattern Trees and Query Plans



Definition

A pattern tree T is a pair T =(T,P), where T = (V, E, r) is a rooted tree, and P =$(P_n)_{n \in V}$ is a labeling of the nodes of T such that Pn is a nonempty set of triple patterns, for every n ∈ V .

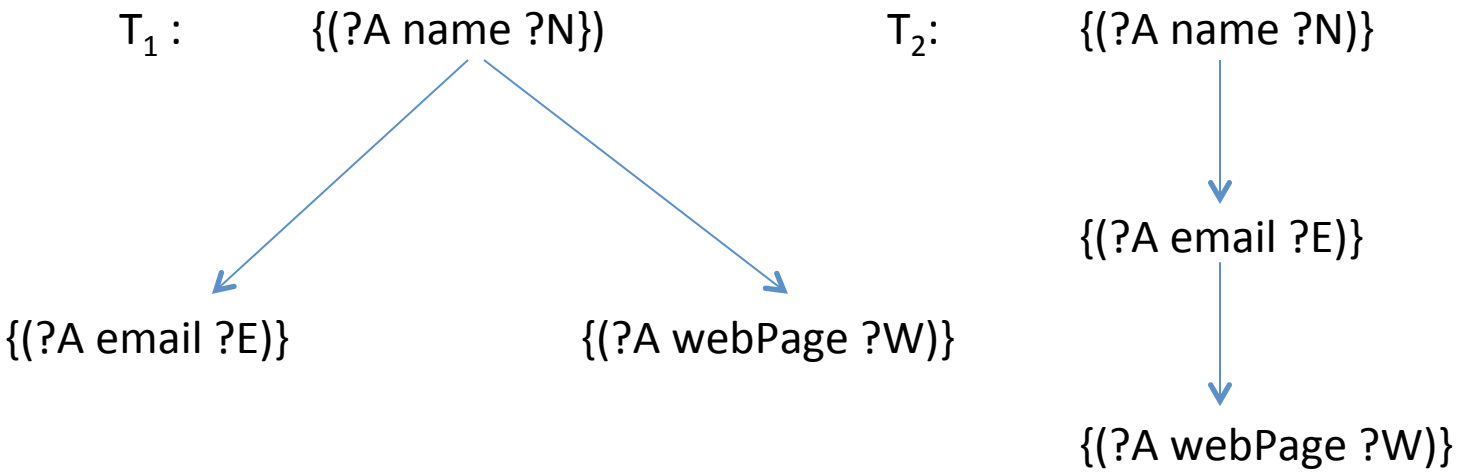
Transform pattern tree to sparql graph pattern

- pattern tree $T = ((V, E, r), P)$
- set Σ of functions $\{\sigma_n \mid n \in V\}$ for every $n \in V$, function σ_n defines an ordering on the children of n
- $P = \{t_1, \ldots, t_\ell\}$
- and(P) the graph pattern $(t_1 \text{ AND } t_2 \text{ AND } \cdots \text{ AND } t_\ell)$

transformation $T_R(T, n, \Sigma)$ of T_n =

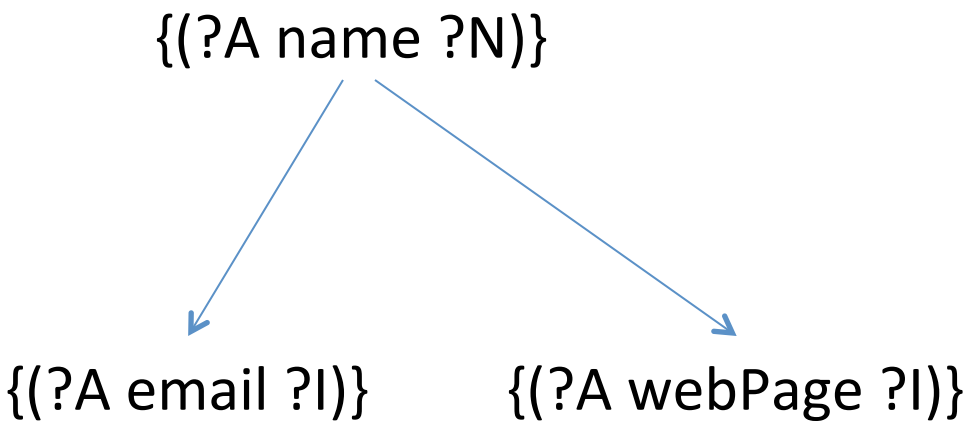
$$\text{and}(P_n) \text{ OPT } T_R(T, \sigma_n(1), \Sigma) \text{ OPT } T_R(T, \sigma_n(2), \Sigma) \ldots \text{ OPT } T_R(T, \sigma_n(k), \Sigma)$$

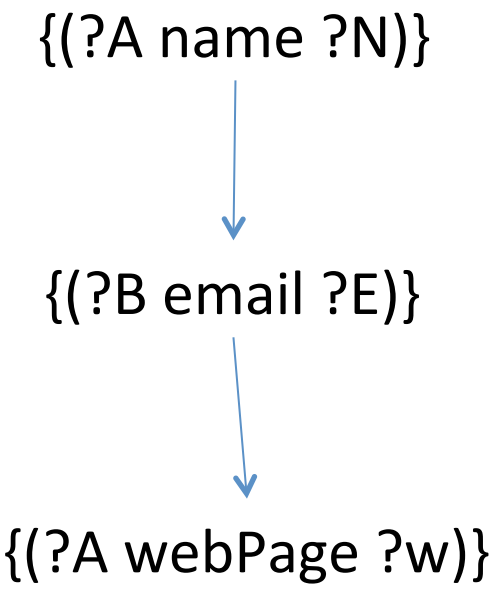
Well-designed graph patterns Trees



A pattern tree is well-designed if for every variable ?X occurring in T, the set {n ϵV | ?X ϵ vars(Pn)} induces a connected sub graph

Not Well-designed pattern tree



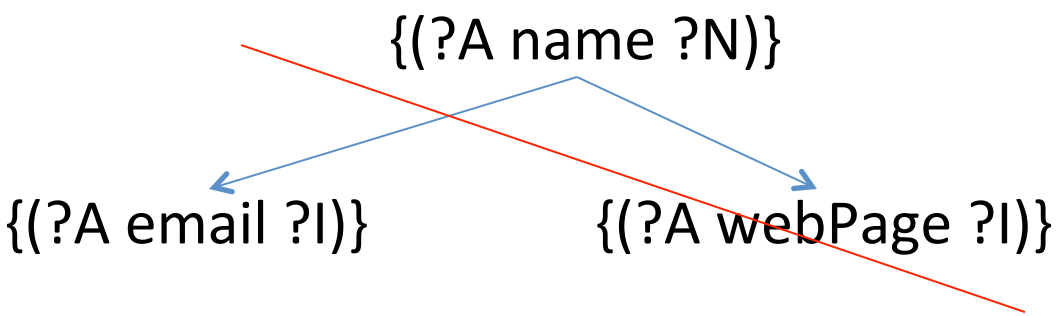


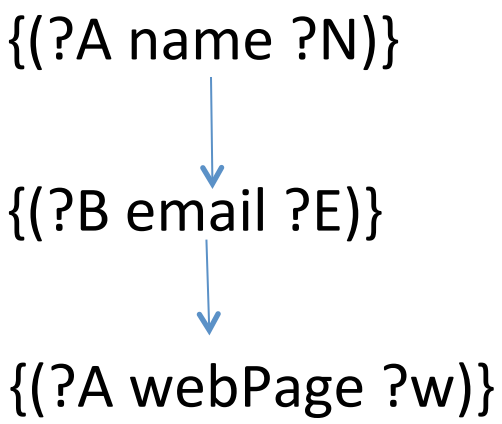
Variable ?I in the tree on the left, and variable ?A in the tree on the right, induce disconnected subgraphs

Quasi well-designed pattern tree

Definition

A pattern tree $T = ((V, E, r),(P_n)_{n \in V})$ is a quasi well-designed pattern tree (QWDPT for short) if for every pair of nodes $u, v \in V$ and each variable $?X \in vars(P_u) \cap vars(P_v)$ there exists a node n that is a common ancestor of u and v in T , such that $?X \in vars(P_n)$





QWDPT and well-designed pattern tree

Every QWDPT can be converted to well designed pattern tree by duplicating triples along branches.

Definition

Let T be a QWDPT. The set of SPARQL graph patterns defined by T is
SEM(T) = {$T_R(T',\Sigma)$ | Σ is an ordering for T', T $\rightarrow$ * T' and T' is well-designed}.

Lemma : Let T be a well-designed pattern tree, let Σ_1, Σ_2 be two arbitrary orderings for T , and let $P_1 = T_R(T,\Sigma_1)$ and $P_2 = T_R(T,\Sigma_2)$ be the graph patterns obtained by transforming T with Σ_1 and Σ_2, respectively. Then $P_1 \equiv P_2$.

Lemma : Let T be a QWDPT, let Σ be an ordering for T , and let T_1 and T_2 be well-designed pattern trees such that $T \overset{\cdot\cdot}{\rightarrow} * T_1$ and $T \overset{\cdot\cdot}{\rightarrow} * T_2$.
If $P_1 = TR(T_1,\Sigma)$ and $P_2 = TR(T_2,\Sigma)$, then $P_1 \equiv P_2$

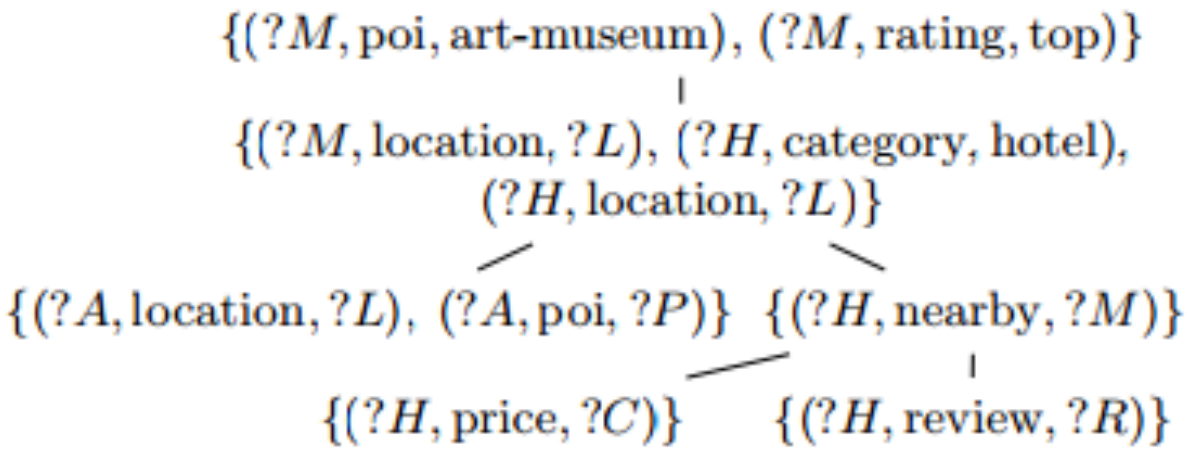
From above 2 lemma,
All graph patterns of QWDPT in SEM(T) are equivalent.

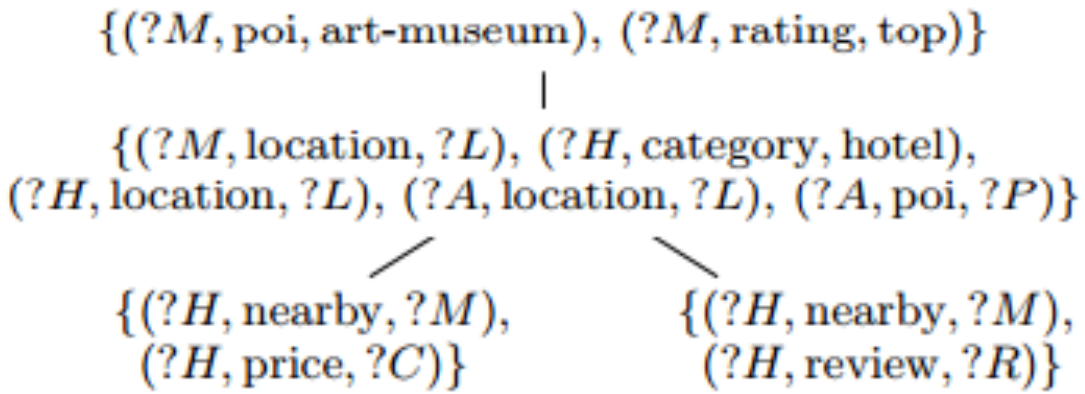
Transformation Of QWDPT

several transformation rules can be applied to pattern trees in order to minimize the number of nodes, and thus, minimizing the number of OPT operators in the query

- R_1: If a triple pattern t belongs to node n and to a descendant n′ of n, then delete t from n′. (deletion of redundant triples)
- R_2: If node n does not introduce any new variable w.r.t. its ancestors, then push copies of n into its children. (deletion of unproductive nodes)
- R_3: If there is a homomorphism from node n to the branch from the root to n, then merge node n with its parent. (homomorphism upwards)
- R_4: If there exists a homomorphism h: $P_n \rightarrow P_{n'} \cup P_{branch(\hat{}n)}$ then turn n′ from a child of n into a child of ^n (parallelization)

Example of Transformation





Example taken from “SPAM: A SPARQL Analysis and Manipulation Tool”

Results from transformation

Theorem:

Let T be a QWDPT and T' the pattern tree that results from applying either rule R_1, or R_2, or R_3, or R_4, to T .Then T' is a QWDPT such that $T \equiv T'$

Let T be a QWDPT. Then the following hold:

1. Iteratively applying rules R_1 and R_2 (in arbitrary order) to T leads to a unique pattern tree T^* in NR normal form.
2. If T is in NR normal form then it remains in NR normal form when applying rules R_3 or R_4 to T
3. T is in R3 normal form if T is reduced w.r.t. rules R1, R2, and R3.

4. Containment

Containment:

determining if the result of one query is included in the result of another for any RDF graph

Example:

P_1 = (?X, n,?Y) and P_2 = (?X, n,?Y) OPT (?X, e,?Z)

graph G = {(a, n, b),(a, e, c)}

Then $[[P_1]]_G$ = {μ ={?X → a, ?Y → b}},

while $[[P2]]_G$ = {μ′ = {?X → a, ?Y →b, ?Z → c}}

Hence $P_1 \not\subseteq P_2$

the answer to P_2 contains strictly more information than that toP_1,

and it is easy to see that for no graph G, pattern P_2 returns fewer bindings than P

Complexity of subsumption

For mappings M_1 and M_2 ,

M_1 is subsumed by M_2, denoted by $M_1 \sqsubseteq M_2$, if for every $\mu_1 \in M_1$ there exists a $\mu_2 \in M_2$ such that $\mu_1 \sqsubseteq \mu_2$.

two QWDPTs T_1 and T_2,

T_1 is subsumed by T_2, denoted by $T_1 \sqsubseteq T_2$, if $[[T_1]]_G \sqsubseteq [[T_2]]_G$ holds for every graph G

QWDPTs T_1 and T_2 with roots r_1 and r_2,respectively.

Then $T_1 \sqsubseteq T_2$ if and only if for every sub tree T'_1 of T_1 rooted at r_1, there exists a subtree T'_2 of T_2 rooted at r_2 s.t.:

$\text{vars}(T_1) \subseteq \text{vars}(T'_2)$,

a homomorphism from the triples in T'_2 to the triples in T'_1

5. Enumeration of well-designed SPARQL

- Given an RDF graph G and a well-designed SPARQL graph pattern P, compute all solution μ

```
Enumerate(t, μ)
 1: cqit := new Iterator(EnumerateCQ(P_t, μ));
 2: while( cqit.hasNext() ){
    // let t_1, ..., t_k be the children of t
 3:     maxi := 0;
 4:     μ_curr := cqit.next();
 5:     for( i = 1 to k ){
 6:         it_i := new Iterator(Enumerate(t_i, μ_curr));
 7:         flag_i := it_i.hasNext();
 8:         if(flag_i){
 9:             μ_i := it_i.next();
10:             maxi := i;
11:         }
12:     }
13:     if( ⋀_{i=1}^{k} ¬flag_i ){
14:         output(μ_curr);
15:         continue;
16:     }
```

```
17:    repeat{
18:       output( μ_curr ∪ ⋃_{1≤i≤k ∧ flag_i = true} μ_i );
19:       continueflag := false;
20:       for( i = maxi downto 1 ){
21:          if(it_i.hasNext()){
22:             μ_i := it_i.next();
23:             continueflag := true;
24:             for( j = i+1 to maxi ){
25:                if( flag_j ) {
26:                   it_j := new Iterator(Enumerate(t_j, μ_curr));
27:                   μ_j := it_j.next();
28:                }
29:             }
30:             i := 0; // leave the for-loop
31:          }
32:       }
33:    } until( ¬continueflag)
34: }
```

Enumeration

- Evaluation algorithm reduces the problem of evaluating all the solution QWDPT to the problem of enumerating all solutions of CQs.
- So enumerating all solutions of QWDPT it can be done in polynomial time as CQs.

Counting of well-designed SPARQL

Given an RDF graph G and well-designed SPARQL pattern P, compute the number of solutions μ

Theorem

The problem of counting all solutions of a QWDPT (and hence, of a well-designed SPARQL graph pattern) is #·coNP complete.

References

- A relational algebra for SPARQL by Richard Cyganiak
- SPAM: A SPARQL Analysis and Manipulation Tool by Andres Letelier, Jorge P ´erez, Reinhard Pichler Sebastian Skritek
- Querying Semantic Web Data with SPARQL by Marcelo Arenas, Jorge Pérez

THANK YOU